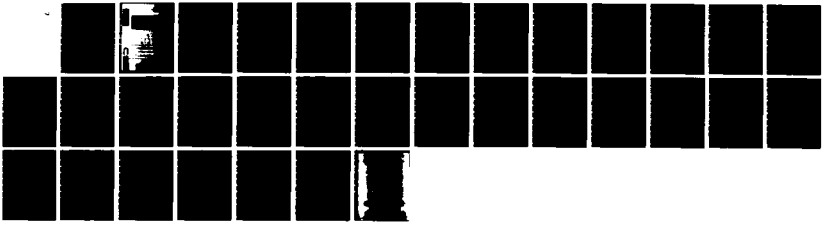
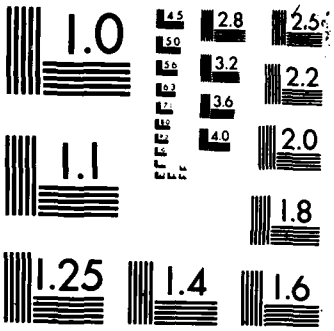
AD-A141 779 PATH RELAXATION: PATH PLANNING FOR A MOBILE ROBOT(U) 1/1
CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST
C E THORPE APR 84 CMU-RI-TR-84-5 N00014-81-K-0503
UNCLASSIFIED F/G 12/1 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

AD-A141 779

Path Relaxation: Path Planning for a Mobile Robot

Charles E. Thorpe

CMU-RI-TR-84-5

Carnegie-Mellon University
The Robotics Institute

Technical Report

THE ROBOTICS INSTITUTE
DTIC FILE COPY

Path Relaxation: Path Planning for a Mobile Robot

Charles E. Thorpe

CMU-RI-TR-84-5

The Robotics Institute
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

April 1984

ELECTE
JUN 5 1984
A

Copyright © 1984 Mobile Robot Laboratory, Carnegie-Mellon University

The CMU Rover has been supported at the Carnegie-Mellon University Robotics Institute since 1981 by the Office of Naval Research under contract number N00014-81-K-0503. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Office of Naval Research or the U. S. Government.

This document has been approved for public release and sale; its distribution is unlimited.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: CMU-RI-TR-84-5	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): Path Relaxation: Path Planning for a Mobile Robot	5. TYPE OF REPORT & PERIOD COVERED: Interim
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Charles E. Thorpe	8. CONTRACT OR GRANT NUMBER(s): ONR N00014-81-K-0503
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Carnegie-Mellon University, The Robotics Institute, Pittsburgh, PA. 15213	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Arlington, VA 22217	12. REPORT DATE: April 1984
	13. NUMBER OF PAGES: 27
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Approved for public release; distribution unlimited

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

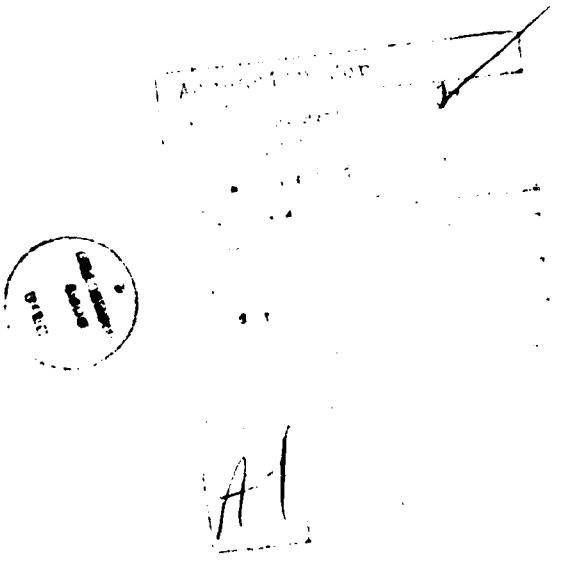
Path Relaxation is a method of planning safe paths around obstacles for mobile robots. It works in two steps: a global grid search that finds a rough path, followed by a local relaxation step that adjusts each node on the path to lower the overall path cost. The representation used by Path Relaxation allows an explicit tradeoff among length of path, clearance away from obstacles, and distance traveled through unmapped areas.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Table of Contents

1. Introduction 1
2. Constraints 2
3. Approaches to Path Planning 4
4. Path Relaxation 6
   4.1 Grid Size 6
   4.2 Grid Search 8
   4.3 Relaxation 10
5. Additions to the Basic Scheme 12
6. Remaining Work 13
Acknowledgements 15
Example Runs 16



List of Figures

Figure 1: To find open spaces on a 1D line big enough to hold a robot of size *r*, it is sufficient to sample the line at intervals of *r*. 6

Figure 2: On a 4-connected 2D grid of size *r*, it is possible to miss diagonal paths. Here, there is enough room for the robot to go from (1,1) to (2,2), but there is no clear path on the grid. 7

Figure 3: One fix for the problem in figure 2 is to shrink the grid by a factor of sqrt(2). Then, if the robot will fit, there will be a grid path. Here the robot could go from (1,1) to (2,1) to (2,2) to (3,2) and on. 7

Figure 4: Another fix is to use an 8-connected grid. Now there is a diagonal line directly connecting (1,1) with (2,2). 8

Abstract

Path Relaxation is a method of planning safe paths around obstacles for mobile robots. It works in two steps: a global grid search that finds a rough path, followed by a local relaxation step that adjusts each node on the path to lower the overall path cost. The representation used by Path Relaxation allows an explicit tradeoff among length of path, clearance away from obstacles, and distance traveled through unmapped areas.

1. Introduction

Path Relaxation is a two-step process for mobile robots. It finds a safe path for a robot to traverse a field of obstacles and arrive at its destination. The first step of path relaxation finds a preliminary path on an eight-connected grid of points. The second step adjusts, or "relaxes", the position of each preliminary path point to improve the path.

One advantage of path relaxation is that it allows many different factors to be considered in choosing a path. Typical path planning algorithms evaluate the cost of alternative paths solely on the basis of path length. The cost function used by Path Relaxation, in contrast, also includes how close the path comes to objects (the further away, the lower the cost) and penalties for traveling through areas out of the field of view. The effect is to produce paths that neither clip the corners of obstacles nor make wide deviations around isolated objects, and that prefer to stay in mapped terrain unless a path through unmapped regions is substantially shorter. Other factors, such as sharpness of corners or visibility of landmarks, could also be added for a particular robot or mission.

Path Relaxation is part of Fido, the vision and navigation system of the CMU Rover mobile robot. [29, 41] The Rover, under Fido's control, will navigate solely by stereo vision. It will pick about 40 points to track, find them in a pair of stereo images, and calculate their 3D positions relative to the Rover. The Rover will then move about half a meter, take a new pair of pictures, find the 40 tracked points in each of the new pictures and recalculate their positions. The change in position of those points relative to the robot gives the actual change of the robot's position in the stationary world.

Fido's model of the world is not suitable for most existing path-planning algorithms. Algorithms for planning paths usually assume a completely known world model composed of planar-faced objects. Fido's world model, on the other hand, contains only the 40 points it is tracking. For each point the model records its position, the uncertainty in that position, and the appearance of a small patch of the image around that point. Furthermore, Fido only knows about what it has seen; points that have never been within its field of view are not listed in the world model. Also, the vision system may fail to track points correctly, so there may be phantom objects in the world model that have been seen once but are no longer being tracked. All this indicates the need for a data structure that can represent uncertainty and inaccuracy, and for algorithms that can use such data.

Section 2 of this report outlines the constraints available to Fido's path planner. Section 3 discusses some common types of path planners, and shows how they are inadequate for our application. The Path Relaxation algorithm is explained in detail in Section 4, and some additions to the basic scheme are presented in Section 5. Finally, Section 6 discusses shortcomings of Path Relaxation and some possible extensions.

2. Constraints

An intelligent path planner needs to bring much information to bear on the problem. This section discusses some of the information useful for mobile robot path planning, and shows how the constraints for mobile robot paths differ from those for manipulator trajectories.

Low dimensionality. A ground-based robot vehicle is constrained to three degrees of freedom: x and y position and orientation. In particular, the CMU Rover has a circular cross-section, so for the purposes of path planning the orientation does not matter. This makes path planning only a two-dimensional problem. In contrast, typical robot arms have from 3 to 6 joints. A path planner for a manipulator, if it takes into account the position of each joint, would have up to a 6-dimensional search space.

Imprecise control. Even under the best of circumstances, a mobile robot is not likely to be very accurate. Its precision will depend on the smoothness of the ground, the accuracy of its controller, and the traction of the wheels. Typical manipulators, on the other hand, have repeatabilities of a few thousandths of an inch over their entire reach. The implication for path planning is that it is much less important to worry about exact fits for mobile robot paths. If the robot could, theoretically, just barely fit through a certain opening, then in practice that's probably not a good way to go. Computational resources are better spent exploring alternate paths rather than worrying about highly accurate motion calculations.

Cumulative error. Errors in a dead-reckoning system tend to accumulate: a small error in heading, for instance, can give rise to a large error in position as the vehicle moves. The only way to reduce error is to periodically measure position against some global standard, which can be time-consuming. The Rover, for example, does its measurement by stereo vision, taking a few minutes to compute its exact position. So a slightly longer path that stays farther away from obstacles, and allows longer motion between stops for measurement, may take less time to travel than a shorter path that requires more frequent stops. In contrast, a manipulator can reach a location with approximately the same error regardless of what path is taken to arrive there. There is no cumulative error, and no time spent in reorientation.

Unknown areas. Robot manipulator trajectory planners usually know about all the obstacles. The Rover knows only about those that it has seen. This leaves unknown areas outside its field of view and behind obstacles. It is usually preferable to plan a path that traverses only known empty regions, but if that path is much longer than the shortest possible path then it may be worth while looking at the unknown regions. Perhaps some "curiosity factor", that changed depending on whether this was a mapping run or a production run, could determine the tendency to look around.

Fuzzy objects. Not only do typical manipulator path-planners know about all the objects, they know precisely where each object is. This information might come, for instance, from the CAD system that designed the robot workstation. Mobile robots, on the other hand, usually sense the world as they go. The Rover, instead of having precise bounds for objects, knows only about fuzzy points. The location of a point is only known to the precision of the stereo vision system, and the extent of an object beyond the point is entirely unknown.

Field of view. It may be important for a mobile robot to plan paths that keep it as far away from obstacles as possible, so that it can see more distant objects. Or it may be important to keep it behind objects, so that it cannot be seen from far off: imagine a robot watchman, or a robot tank, sneaking up on some hostile force.

In summary, a good system for mobile robot path planning will be quite different from a manipulator path planner. Mobile robot path planners need to handle uncertainty in the sensed world model and errors in path execution. They do not have to worry about high dimensionality or extremely high accuracy. Section 3 of this report discusses some existing path planning algorithms and their shortcomings. Section 4 then presents the algorithms used by Path Relaxation, and shows how they address these problems.

3. Approaches to Path Planning

This section outlines several approaches to path planning and some of the drawbacks of each approach. All of these methods except the potential fields approach abstract the search space to a graph of possible paths. This graph is then searched by some standard search technique, such as breadth-first or A* [30, 38, 39], and the shortest path is returned. The important thing to note in the following is the information made explicit by each representation and the information thrown away.

Free Space methods. [3, 13, 31, 32] One type of path planner explicitly deals with the free space rather than with the space occupied by obstacles, and forces path segments to run down the middle of the corridors between obstacles. One such method fits generalized cylinders to the free space. Another calculates the free space's Voronoi diagram. The spine of the Voronoi diagram or the axes of the generalized cylinders form a network of possible paths. Some of these paths may pass through places narrower than the robot: these path segments can be detected and deleted. The remaining segments form the graph of possible paths which is searched to find the shortest path.

Free space algorithms suffer from two related problems, both resulting from a data abstraction that throws away too much information. The first problem is that paths always run down the middle of corridors. In a narrow space, this is desirable, since it allows the maximum possible robot error without hitting an object. But in some cases paths may go much further out of their way than necessary. The second problem is that the algorithms do not use clearance information. The shortest path is always selected, even if it involves much closer tolerances than a slightly longer path.

Vertex Graphs. [24, 40, 28] Another class of algorithms is based on a graph connecting pairs of vertices. In the first step, each obstacle is expanded by the size of the robot and the robot conceptually shrunk to a point. The problem of finding a path for the point through the grown obstacles is exactly the same as finding a path for the whole robot through the original objects. The graph of possible paths is built by considering every pair of vertices of the expanded obstacles: if the line between two vertices does not intersect any of the expanded obstacles, it is a candidate path segment and is added to the graph. As in the Free Space methods, the graph is searched by some standard graph search algorithm, and the shortest path is returned. Variations of this algorithm use schemes that interleave generation and testing of the graph, hoping to avoid building parts of the graph.

Vertex graph algorithms suffer from the "too close" problem: in their concern for the shortest possible path, they find paths that clip the corners of obstacles and even run along the edges of some objects. It is, of course, possible to build in a margin of error by growing the obstacles by an extra amount; this may, however, block some paths.

Both free space and vertex graph methods throw away too much information too soon. All obstacles are modeled as polygons, all paths are considered either open or blocked, and the shortest path is always best. There is no mechanism for trading a slightly longer path for more clearance, or for making local path adjustments. There is also no clean way to deal with unmapped regions, other than to close them off entirely.

The **Potential Fields** [1, 20] approach tries to make those tradeoffs explicit. Conceptually, it turns the robot into a marble, tilts the floor towards the goal, and watches to see which way the marble rolls. Obstacles are represented as hills with sloping sides, so the marble will roll a prudent distance away from them but not too far, and will seek the passes between adjacent hills. Sophisticated algorithms even give the marble momentum, taking into account the energy needed to accelerate, decelerate, and turn. The problem with potential field paths is that they tend to get caught in dead ends: once the marble rolls into a box canyon, the algorithm has to invoke special-case mechanisms to cut off that route, backtrack, and start again. Moreover, the path with the lowest threshold might turn out to be a long and winding road, while a path that climb a small ridge at the start and then has an easy run to the goal might never be investigated. So of these problems can be avoided if there are no concave objects or collections of objects, but this is a rong and unrealistic restriction.

Another approach that could explicitly represent the conflicts between short paths and obstacle avoidance is the **Regular Grid** method. This covers the world with a regular grid of points, each connected with its 4 or 8 neighbors to form a graph. In existing regular grid implementations, the only information stored at a node is whether it is inside an object or not. Then the graph is searched, and the shortest grid path returned. This straightforward grid search has many of the same "too close" problems as the vertex graph approaches. A more sophisticated approach could assign weights to the nodes depending on how close they were to objects or other factors, and avoid the "too close" and "too far" problems. But the remaining path would still be jagged, and might miss the best path because of the grid's coarse resolution.

4. Path Relaxation

Path Relaxation combines the best features of grid search and potential fields. Using the rolling marble analogy, the first step is a global grid search that finds a good valley for the path to follow. The second step is a local relaxation step, similar to the potential field approach, that moves the nodes in the path to the bottom of the valley in which they lie. The terrain (cost function) consists of a gradual slope towards the goal, hills with sloping sides for obstacles, and plateaus for unexplored regions. The height of the hills has to do with the confidence that there really is an object there. Hill diameter depends on robot precision: a more precise robot can drive closer to an object, so the hills will be tall and narrow, while a less accurate vehicle will need more clearance, requiring wide, gradually tapering hillsides.

This section first presents results on how large the grid size can be without missing paths. It next discusses the mechanism for assigning cost to the nodes and searching the grid. Finally, it presents the relaxation step that adjusts the positions of path nodes.

4.1 Grid Size

How large can a grid be and still not miss any possible paths? That depends on the number of dimensions of the problem, on the connectivity of the grid, and on the size of the vehicle. It also depends on the vehicle's shape: in this section, we discuss the simplest shape, which is a vehicle with a circular cross-section.

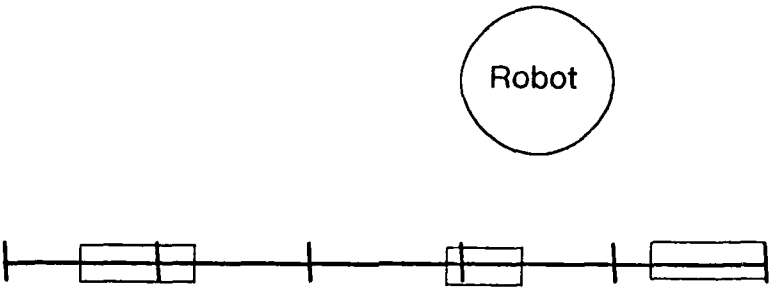


Figure 1: To find open spaces on a 1D line big enough to hold a robot of size r, it is sufficient to sample the line at intervals of r.

First consider the problem of placing a circle on a 1D line (see figure 1). Given a line segment with some sections covered with 1D obstacles, and a robot of size r, we wish to find all clear places on the line where the robot can be placed. It is sufficient in this case to sample the segment at intervals of r and test only the unblocked points. A 1D grid spacing of r guarantees that any open section of length r or greater will touch at least one of the grid points. An open point does not guarantee that it is part of an open interval of size r; it merely says that point is worth investigating.

We can extend this reasoning from one dimension to a more useful case, a 2D area with 2D obstacles. The area can be covered with a grid in which each node is connected to either its four or its eight nearest neighbors. For a four-connected grid, if the spacing were r, there would be a chance of missing diagonal

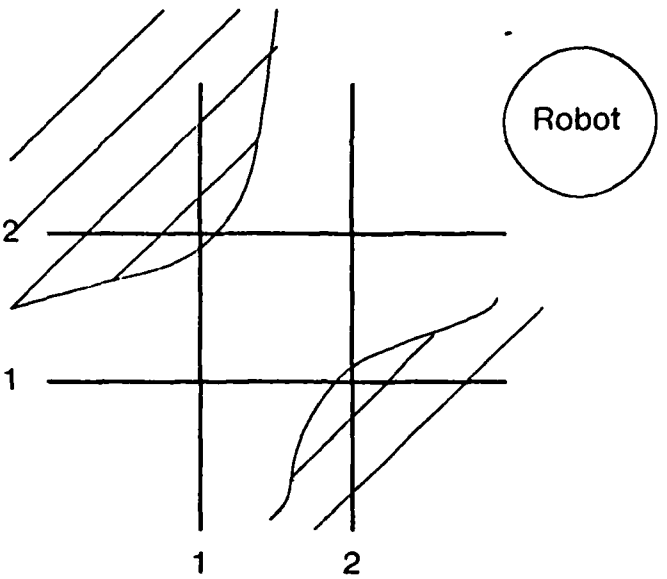


Figure 2: On a 4-connected 2D grid of size r, it is possible to miss diagonal paths. Here, there is enough room for the robot to go from (1,1) to (2,2), but there is no clear path on the grid.

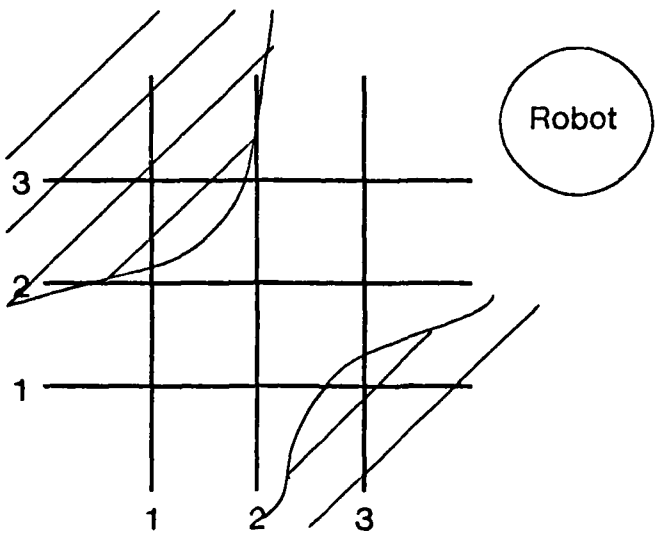


Figure 3: One fix for the problem in figure 2 is to shrink the grid by a factor of sqrt(2). Then, if the robot will fit, there will be a grid path. Here the robot could go from (1,1) to (2,1) to (2,2) to (3,2) and on.

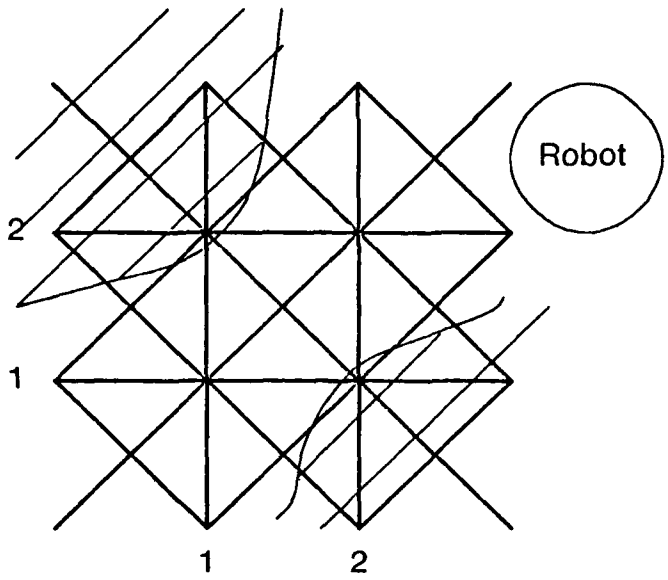


Figure 4: Another fix is to use an 8-connected grid. Now there is a diagonal line directly connecting (1,1) with (2,2).

paths: there might be just enough room between two obstacles for the robot to move from (x, y) to (x+1, y+1), yet both node (x, y+1) and node (x+1, y) might be covered. To guarantee that no paths are missed, the grid spacing must be reduced to $r\sqrt{2}/2$. That is the largest size allowable that guarantees that if diagonally opposite nodes are covered, there is not enough room between them for the robot to safely pass.

If the grid is eight-connected (each node connected to its diagonal, as well as orthogonal, neighbors), the problem with diagonal paths disappears. As in the 1D case, the grid spacing can be a full r, while guaranteeing that if there is a path it will be found.

4.2 Grid Search

Once the grid size has been fixed, the next step is to assign costs to paths on the grid and then to search for the best path along the grid from the start to the goal. "Best", in this case, has three conflicting requirements: shorter path length, greater margin away from obstacles, and less distance in uncharted areas. These three are explicitly balanced by the way path costs are calculated. A path's cost is the sum of the costs of the nodes through which it passes, each multiplied by the distance to the adjacent nodes. (In a 4-connected graph all lengths are the same, but in an 8-connected graph we have to distinguish between orthogonal and diagonal links.) The node costs consist of three parts to explicitly represent the three conflicting criteria.

1. Cost for distance. Each node starts out with a cost of one unit, for length traveled.

2. Cost for near objects. Each object near a node adds to that node's cost. The nearer the obstacle, the more cost it adds. The exact slope of the cost function will depend on the accuracy of the vehicle (a more accurate vehicle can afford to come closer to objects), and the vehicle's speed (a faster vehicle can afford to go farther out of its way), among other factors.

3. Cost for within or near an unmapped region. The cost for traveling in an unmapped region will depend on the vehicle's mission. If this is primarily an exploration trip, for example, the cost might be relatively low. There is also a cost added for being near an unmapped region, using the same sort of function of distance as is used for obstacles. This provides a buffer to keep paths from coming too close to potentially unmapped hazards.

The first step of Path Relaxation is to set up the grid and construct the list of obstacles and the vehicle's current position and field of view.[1] The system can then calculate the cost at each node, based on the distances to nearby obstacles and whether that node is within the field of view or not. The next step is to create links from each node to its 8 neighbors. The start and goal locations do not necessarily lie on grid points, so special nodes need to be created for them and linked into the graph.

The system then searches this graph for the minimum-cost path from the start to the goal. The search itself is a standard A* [30] search. The estimated total cost of a path, used by A* to pick which node to expand next, is the sum of the cost so far plus the straight-line distance from the current location to the goal. This has the effect, in regions of equal cost, of finding the path that most closely approximates the straight-line path to the goal.

The path found is guaranteed to be the lowest-cost path on the grid, but this is not necessarily the overall optimal path. First of all, even in areas with no obstacles the grid path may be longer than a straight-line path simply because it has to follow grid lines. For a 4-connected grid, the worst case is diagonal lines, where the grid path is $\sqrt{2}$ times as long as the straight-line path. For an 8-connected grid, the equivalent worst case is a path that goes equal distances forward and diagonally. This gives a path about 1.08 times as long as the straight-line path. In cases where the path curves around several obstacles, the extra path length can be even more significant. Secondly, if the grid path goes between two obstacles, it may be non-optimal because a node is placed closer to one obstacle than to the other. A node placed exactly half way between the two obstacles would, for most types of cost functions, have a lower cost. The placement of the node that minimizes the overall path cost will depend both on node cost and on path length, but in any case is unlikely to be exactly on a grid point. If the grid path is topologically equivalent to the optimal path (i.e. goes on the same side of each object), the grid path can be iteratively improved to approximate the optimal path (see Section 5). But if the grid path at any point goes on the "wrong" side of an obstacle, then no amount of local adjustment will yield the optimal path. The chance of going on the wrong side of an obstacle is related to the size of the grid and the shape of the cost vs. distance function. For a given grid size and cost function, it is possible to put a limit on how much worse the path found could possibly be than the optimal path. If the result is too imprecise, the grid size can be decreased until the additional computation time is no longer worth the improved path.

Up to this point, it has been assumed that the path is passable, that is that there is enough clearance for the vehicle at all points. At this point that assumption can be tested. If any path link goes between two objects

---

[1] In this implementation, there are two types of obstacles: polygonal and circular. Currently, the circular obstacles are used for points found by Fido's vision system, each bounded by a circular error limit, and the polygons are used for the field of view. The vision system will eventually give polygonal obstacles, at which point both the obstacles and the field of view will be represented as polygons and the circular obstacles will no longer be needed.

separated by less than the robot diameter, the robot cannot squeeze through that gap and the path is not passable. The link that was crossed can be removed, and the grid search redone. In practice, if the best grid path is not passable, it is likely that no other path will be, either. However it is possible for the lowest-cost path to have a single spot that is just barely impassable, while the next lowest cost path has a series of just barely passable spots that add up to higher total cost. The probability of this happening depends on the shape of the cost function, the size of the grid, the length of the path, and the amount of clutter present.

A few details on the shape of the cost function deserve mention. Many different cost functions will work, but some shapes are harder to handle properly. The first shape we tried was linear. This had the advantage of being easy to calculate quickly, but gave problems when two objects were close together. The sum of the costs from two nearby objects was equal to a linear function of the sum of the distances to the objects. This creates ellipses of equal cost, including the degenerate ellipse on the line between the two objects. In that case, there was no reason for the path to pick a spot midway between the objects, as we had (incorrectly) expected. Instead, the only change in cost came from changing distance, so the path went wherever it had to to minimize path length. In our first attempt to remedy the situation we replaced the linear slope with an exponentially decaying value. This had the desired effect of creating a saddle between the two peaks, and forcing the path towards the midpoint between the objects. The problem with exponentials is that they never reach zero. For a linear function, there was a quick test to see if a given object was close enough to a given point to have any influence. If it was too far away, the function did not have to be evaluated. For the exponential cost function, on the other hand, the cost function had to be calculated for every obstacle for each point. We tried cutting off the size of the exponential, but this left a small ridge at the extremum of the function, and paths tended to run in nice circular arcs along those ridges. A good compromise, and the function we finally settled on, is a cubic function that ranges from 0 at some maximum distance, set by the user, to the obstacle's maximum cost at 0 distance. This has both the advantages of having a good saddle between neighboring obstacles and of being easy to compute and bounded in a local area.

4.3 Relaxation

Grid search finds an approximate path; the next step is an optimization step that fine-tunes the location of each node on the path to minimize the total cost. One way to do this would be to precisely define the cost of the path by a set of non-linear equations and solve them simultaneously to analytically determine the optimal position of each node. This approach is not, in general, computationally feasible. The approach used here is a relaxation method. Each node's position is adjusted in turn, using only local information to minimize the cost of the path sections on either side of that node. Since moving one node may affect the cost of its neighbors, the entire procedure is repeated until no node moves farther than some small amount.

Node motion has to be restricted. If nodes were allowed to move in any direction, they would all end up at low cost points, with many nodes bunched together and a few long links between them. This would not give a very good picture of the actual cost along the path. So in order to keep the nodes spread out, a node's motion is restricted to be perpendicular to a line between the preceding and following nodes. Furthermore, at any one step a node is allowed to move no more than one unit.

As a node moves, all three factors of cost are affected: distance traveled (from the preceding node, via this

node, to the next node), proximity to objects, and relationship to unmapped regions. The combination of these factors makes it difficult to directly solve for minimum cost node position. Instead, a binary search is used to find that position to whatever accuracy is desired.

The relaxation step has the effect of turning jagged lines into straight ones where possible, of finding the "saddle" in the cost function between two objects, and of curving around isolated objects. It also does the "right thing" at region boundaries. The least cost path crossing a border between different cost regions will follow the same path as a ray of light refracting at a boundary between media with different transmission velocities. The relaxed path will approach that path.

5. Additions to the Basic Scheme

There are several useful additions to the basic path relaxation approach. This section describes those additions that have been implemented and tested. Section 6 describes other possible extensions.

One extension is to vary the costs of individual obstacles. The vision system is not always accurate; it sometimes reports phantom objects that do not really exist, and sometimes loses real objects that it had tracked correctly for several steps. If the path planner kept all the objects the vision system ever thought it saw, the world map could get hopelessly jammed. On the other hand, if an object were only used for path planning when it was currently being tracked, the robot could collide with objects that had been picked up by the vision system at one point. The solution is to decrease the cost of objects that are within the field of view and should be visible, but are not reported by the vision system. If the cost goes below zero, that object gets deleted. Typically, the cost is decreased by 10% of its original value for each turn it is within the field of view but not seen. So 10 turns after the vision system loses an object, it disappears from the path planner's world model. This is a cost-effective, but not very sophisticated approach to the problem. As the vision system improves and gives more reliable data, this will become a more interesting problem and will require more work.

Another extension implemented is to re-use existing paths whenever possible. Fido invokes the path planner at the beginning of each step. The planner adds new points to its world map, adjusts costs of existing points, and calculates the current field of view. It then takes the previously calculated path, trims off nodes that the vehicle has moved through, and splices in the current position as the new start node. It checks this path to see if it is still usable; each node and each line between nodes is checked against all obstacles to see if it comes close enough to cause a collision. If the entire path is clear, it is used again for the next step. Only if it is blocked at some point does the path-planner have to start from scratch. A more sophisticated scheme would also consider the chance that a new, shorter path had opened up because of deleted obstacles or new area in the field of view. It would then estimate the amount of time that could be saved by finding a new path, and the chance that a new path could be found, before it decided whether to reuse the existing path or to replan from scratch.

The relaxation step can be greatly speeded up if it runs in parallel on several computers. Although an actual parallel implementation has not yet been done, a simulation has been written and tested. The major difference is that by the time a node's position is to be adjusted in the serial implementation, the previous node's position has already been updated, so the adjustments ripple down the chain of nodes. If all nodes are to be adjusted in parallel, though, a node's position has to updated based on its predecessor's current, rather than new, position. One effect of this is that changes may take more iterations to propagate down the chain. Another effect is that certain instabilities can occur. For example, a zig-zagging path can flip from the original path to its mirror image on alternate iterations. This can be fixed with appropriate damping.

6. Remaining Work

Path Relaxation would be easy to extend to higher dimensions. It could be used, for example, for a three-dimensional search to be used by underwater vehicles maneuvering through a drilling platform. Another use for higher-dimensional searches would be to include rotations for asymmetric vehicles. Yet another application would be to model moving obstacles; then the third dimension becomes time, with the cost of a grid point having to do with distance to all objects at that time. This would have a slightly different flavor than the other higher-dimensional extensions; it is possible to go both directions in x, y, z, and theta, but only one direction in the time dimension.

Another possible extension has to do with smoothing out sharp corners. All wheels on the CMU Rover steer, so it can follow a path with sharp corners if necessary. Many other vehicles are not so maneuverable; they may steer like a car, with a minimum possible turning radius. In order to accommodate those vehicles, it would be necessary to restrict both the graph search and relaxation steps. A related problem is to use a smoothly curved path rather than a series of linear segments.

Occlusions could be taken into account in fading out objects. The current rules decrease the weight of an object if it has not been seen on the current step, but is within the field of view of the camera. Sometimes an object may not be seen because it is behind a nearer object. So a better fading algorithm would take into account nearby objects that may be blocking further points from view. This strategy becomes much more interesting with a more sophisticated vision system, for example one that reports surfaces rather than points.

In the current system, the error function is the same for each object. One possible extension is to increase the size of obstacles further away from the start position, reflecting the increased positional error of the vehicle as it moves. This has not been done for the Fido/Rover system, since a typical move will probably be less than a meter long. But other systems, interested in longer moves, may want to increase their margin of error along the path. Another possibility would be to use a constant error function, but to determine the distance to be traveled in one step by seeing how far along the current path the vehicle can move before its worst-case position error exceeds the clearance available.

Another area that will require more work is time vs. accuracy trade-offs. Some algorithms have a time complexity that depends only on the number of objects or the number of object vertices. The run-time of path relaxation, in contrast, depends on the size of the area to be covered and on the grid spacing. So situations with lots of empty space and only a few large objects favor other algorithms. Part of that can be overcome with a strategy that automatically uses a larger grid size in clear areas. This brings up problems of representation of non-homogeneous grids and of deciding when to switch grid size. Neither of these appears to be theoretically difficult, but both will take some thought to do elegantly.

A related problem is use of path relaxation for paths with tight clearances. If it is really necessary to squeeze between two obstacles, the current algorithm may not provide a detailed enough path. If multiple grid sizes are implemented, it will be possible to get smaller, as well as larger spacings, and to find good paths even in tight spots.

Path relaxation, as well as almost all existing path planners, deals only with geometric information. A large part of a robot's world knowledge, however, may be in partially symbolic form. For example, a map assembled by the vehicle itself may have very precise local patches, each measured from one robot location. The relations between patches, though, will probably be much less precise, since they depend on robot motion from one step to the next. Using such a mixture of constraints is a hard problem.

Acknowledgements

Many thanks to Hans Moravec, Larry Matthies, and Richard Wallace, all of the CMU Rover Project, for conversations, advice, and encouragement. In particular, Larry is the co-author of Fido; he also gets credit for suggesting the name "Path Relaxation". Richard suggested the names "too-near" and "too-far" used to describe problems with various other path-planning algorithms, and let me use some of his Lisp code for producing the drawings. Larry, Richard, and Hans all made helpful comments on early versions of the manuscript. Thanks also to Nancy Serviou for stylistic comments.

Example Runs

Figures 5, 6, and 7 form a set. Figure 5 is a run from scratch, using real data extracted from the stored images by the Fido vision and navigation system. Objects are represented as little circles, where the size of the circle is the positional uncertainty of the stereo system. The numbers are not all consecutive, because some of the points being tracked are on the floor or are high off the ground, and therefore aren't obstacles. The dotted lines surround the area not in the field of view; this should extend to negative infinity. The start position of the robot is approximately (0, -.2) and the goal is (0, 14.5). The grid path found is marked by 0's. After one iteration of relaxation, the path is marked by 1's. After the second relaxation, the path is marked by 2's. The greatest change from 1 to 2 was less than .3 meters, the threshold, so the process stopped. The size of the "hills" in the cost function is 1 meter, which means that the robot will try to stay 1 meter away from obstacles unless that causes it to go too far out of its way.

After the robot's first move, it tries to reuse the existing path. The robot is now at about (0.1, 0.8). It deletes the old start node and the nodes at (0,0) and (0,1), and links its current position into the path. In this case it had no new objects to add, so the path was still passable. It then uses the old path without having to plan from scratch.

After the next step, there are some new objects added. In particular, object #61 is too close to the old path, so a new one has to be planned. Again, the 0's are the grid path, and the 1's and 2's mark steps of the relaxation algorithm.

Figures 8, 9, and 10 are the same run as shown in figures 5, 6, and 7, except that the relaxation step is simulated to be in parallel. The biggest difference is near the end of the path in figure 8, where the path is somewhat jagged. The path flips between one state on even-numbered iterations (0 and 2) and the opposite state on odd-numbered steps (1). This could be fixed with damping, at the expense of slower convergence on other parts of the path.

References

[1] J. Randolph Andrews.
Impedance Control as a Framework for Implementing Obstacle Avoidance in a Manipulator.
Master's thesis, MIT, 1983.

[2] G. Bauzil, M. Briot, and P. Ribes.
A Navigation Sub-System Using Ultrasonic Sensors for the Mobile Robot HILARE.
In *1st International Conference on Robot Vision and Sensory Controls*. April, 1981.

[3] Rodney Brooks.
Solving the Find-Path Problem by Representing Free Space as Generalized Cones.
AI Memo 674, Massachusetts Institute of Technology, May, 1982.

[4] Rodney Brooks.
Planning Collision Free Motions for Pick and Place Operations.
AI Memo 719, Massachusetts Institute of Technology, May, 1983.

[5] Rodney Brooks and Thomas Lozano-Perez.
A Subdivision Algorithm Configuration Space for Findpath with Rotation.
In *IJCAI-83*. August, 1983.

[6] B. Bullock, D. Keirsey, J. Mitchell, T. Nussmeier, D. Tseng.
Autonomous Vehicle Control: An Overview of the Hughes Project.
In *Trends and Applications 1983*. IEEE Computer Society Press, May, 1983.

[7] Raja Chatila.
Path Finding and Environment Learning in a Mobile Robot System.
In *European Conference on AI*. Orsay, France, July, 1982.

[8] R. Chattergy.
Some Heuristics for the Navigation of a Robot.
Technical Report, University of Hawaii at Manoa, 1983.

[9] Bruno Dufay and Jean-Claude Latombe.
Robot Programming by Inductive Learning.
In *IJCAI-83*. August, 1983.

[10] Pierre Dufresne.
Representation and Processing of Production Rule Interactions in a Robot Planning System.
In *IFAC Symposium on AI*. Lenningrad, USSR, October, 1983.

[11] M. Ferrer, M. Briot, and J. C. Talou.
Study of a Video Image Treatment System for the Mobile Robot HILARE.
In *1st International Conference on Robot Vision and Sensory Controls*. April, 1981.

[12] Richard Fikes, Peter Hart, and Nils Nilsson.
Learning and Executing Generalized Robot Plans.
Technical Report 70, Stanford Research Institute, July, 1972.

[13] Georges Giralt, Ralph Sobek, and Raja Chatila.
A Multi-Level Planning and Navigation System for a Mobile Robot: A First Approach to Hilare.
In *Proceedings of IJCAI-6*. August, 1979.

[14] Georges Giralt, Raja Chatila, and Marc Vaisset.
An Integrated Navigation and Motion Control System for Autonomous Multisensory Mobile Robots.
In *1st International Symposium of Robotic Research*. Bretton Woods, NH, September, 1983.

[15] Eugene Grechanovshk and I. S. Pinsker.
An Algorithm for Moving a Computer-Controlled Manipulator While Avoiding Obstacles.
In *IJCAI-83*. August, 1983.

[16] Leo Guibas, Lyle Ramshaw, and Jorge Stolfi.
A Kinetic Framework for Computational Geometry.
In *Foundations of Computer Science*. IEEE, 1983.

[17] John Hallam.
Resolving Observer Motion by Object Tracking.
In *IJCAI-83*. August, 1983.

[18] J. Hopcroft, J. Schwartz, and M. Sharir.
Efficient Detection of Intersections Among Spheres.
Technical Report 59, Courant Institute, February, 1983.

[19] Scott Kauffman.
An Algorithmic Approach to Intelligent Robot Mobility.
Robotics Age, May/June, 1983.

[20] Bruce Krogh.
Feedback Obstacle Avoidance Control.
1983.
The author is with the Department of Electrical Engineering, Carnegie-Mellon University.

[21] R. E. Larson and A. J. Korsak.
A Dynamic Programming Successive Approximation Technique with Convergence Proofs, Parts I and II.
Automatica 6, 1970.

[22] Jean-Paul Laumond.
Model Structuring and Concept Recognition: Two Aspects of Learning for a Mobile Robot.
In *IJCAI-83*. 1983.

[23] Thomas Lozano-Perez, Matthew Mason, and Russell Taylor.
Automatic Synthesis of Fine-Motion Strategiew for Robots.
AI Memo, Massachusetts Institute of Technology, August, 1983.

[24] Tomas Lozano-Perez and Michael A. Wesley.
An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles.
CACM 22(10), October, 1979.

[25] Matthew Mason.
Automatic Planning of Fine Motions: Correctness and Completeness.
Technical Report, Carnegie-Mellon University, October, 1983.

[26] J. A. Miller.
Autonomous Guidance and Control of a Roving Robot.
In *Proceedings of IJCAI-5*. 1977.

[27] David Miller.
Robot Travel: Planning for Errors.
1984.
Submitted to CSCSI-84. The author is at the Computer Science Department, Yale University.

[28] Hans Moravec.
Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover.
Technical Report CMU-RI-TR-3, Carnegie-Mellon Univesity Robotics Institute, September, 1980.

[29] Hans Moravec.
The CMU Rover.
In *Proceedings of AAAI-82.* August, 1982.

[30] N. Nilsson.
Problem Solving Methods in Artificial Intelligence.
McGraw-Hill, 1971.

[31] Colm O'Dunlaing, Micha Sharir, and Chee Yap.
Retraction: a new approach to motion-planning.
Technical Report, Courant Institute, November, 1982.

[32] C. O'Dunlaing and C. Yap.
The Voronoi Method For Motion-Planning: I. The Case Of A Disc.
Technical Report 53, Courant Institute, March, 1983.

[33] R. S. Rosenberg and P. F. Rowat.
Spatial Problems for a Simulated Robot.
In *IJCAI-81.* August, 1981.

[34] Peter Rowat.
Representing Spatial Experience and Solving Spatial Problems in a Simulated Robot Environment.
PhD thesis, University of British Columbia, October, 1979.

[35] Jacob Schwartz and Micha Sharir.
On the 'Piano Movers' Problem I. The case of a Two-dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers.
Technical Report 39, Courant Institute, October, 1981.

[36] Micha Sharir.
Intersection and Closest-pair Problems for a Set of Planar Objects.
Technical Report 56, Courant Institute, February, 1983.

[37] Paul Spirakis and Chee Yap.
On The Combinational Complexity Of Motion Coordination.
Technical Report 76, Courant Institute, April, 1983.

[38] Robert Tarjan.
A Unified Approach to Path Problems.
JACM 28(3), July, 1981.

[39] Robert Tarjan.
Fast Algorithms for Solving Path Problems.
JACM 28(3), July, 1981.

[40] Alan M. Thompson.
The Navigation System of the JPL Robot.
In *Proceedings of IJCAI-5*. 1977.

[41] Charles Thorpe.
The CMU Rover and the FIDO Vision and Navigation System.
In *Third Symposium on Autonomous Underwater Vehicles*. Marine Systems Engineering Laboratory, University of New Hampshire, 1983.

[42] Charles Thorpe.
An Analysis of Interest Operators for FIDO.
Technical Report CMU-RI-TR-83-19, Robotics Institute, Carnegie-Mellon University, December, 1983.

[43] Shriram M. Udupa.
Collision Detection and Avoidance in Computer Controlled Manipulators.
In *Proceedings of IJCAI-5*. 1977.

[44] Richard Wallace.
Two-Dimensional Path Planning and Collision Avoidance for Three-Dimensional Robot Manipulators.
In *Representation and Processing of Spatial Knowledge*. University of Maryland, May, 1983.

[45] C. Witkowski.
A Parallel Processor Algorithm for Robot Route Planning.
In *IJCAI-83*. August, 1983.

[46] Chee Yap.
Motion Coordination for Two Discs.
Technical Report, Courant Institute, January, 1982.

Figure 5

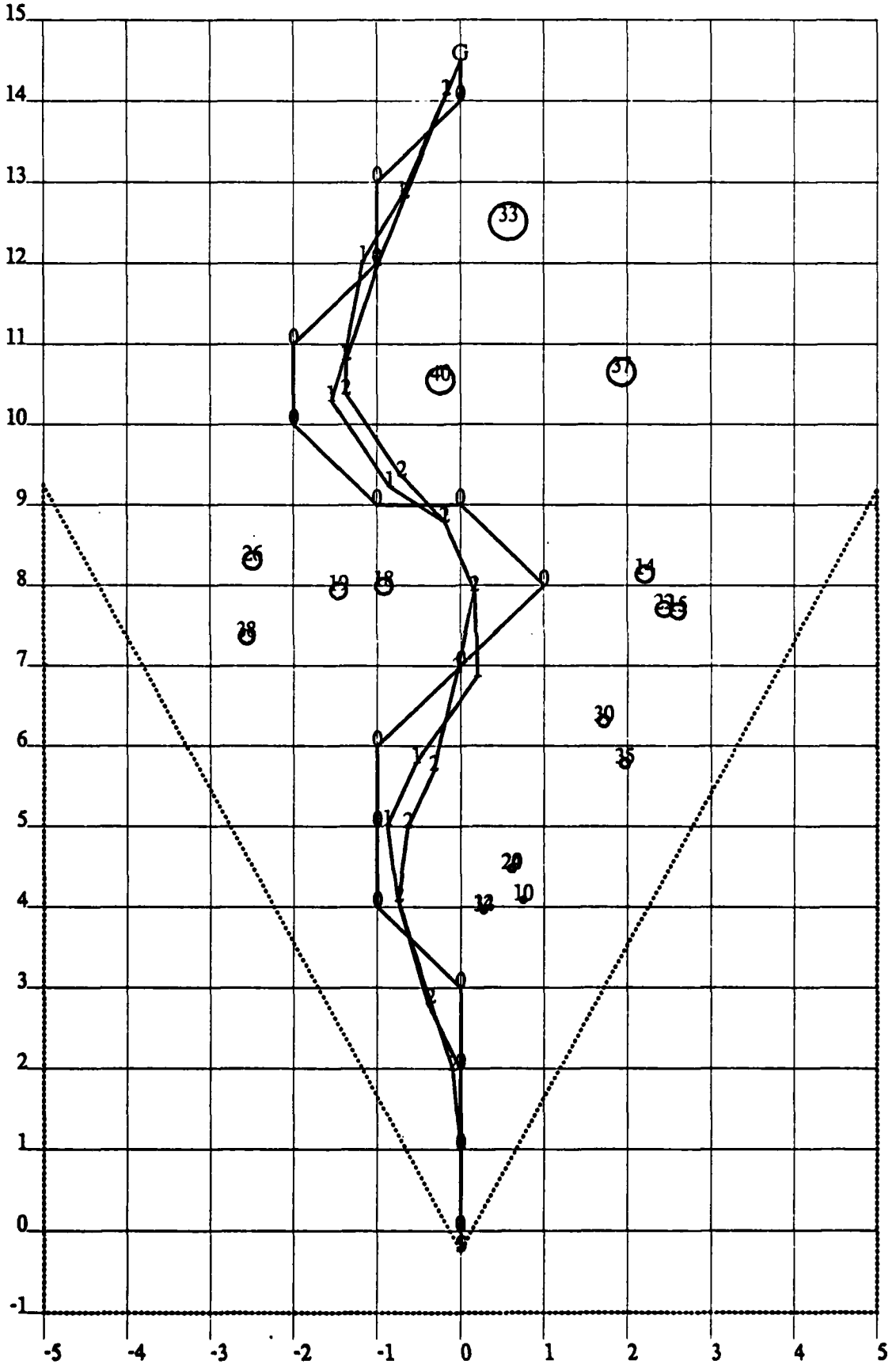


Figure 6

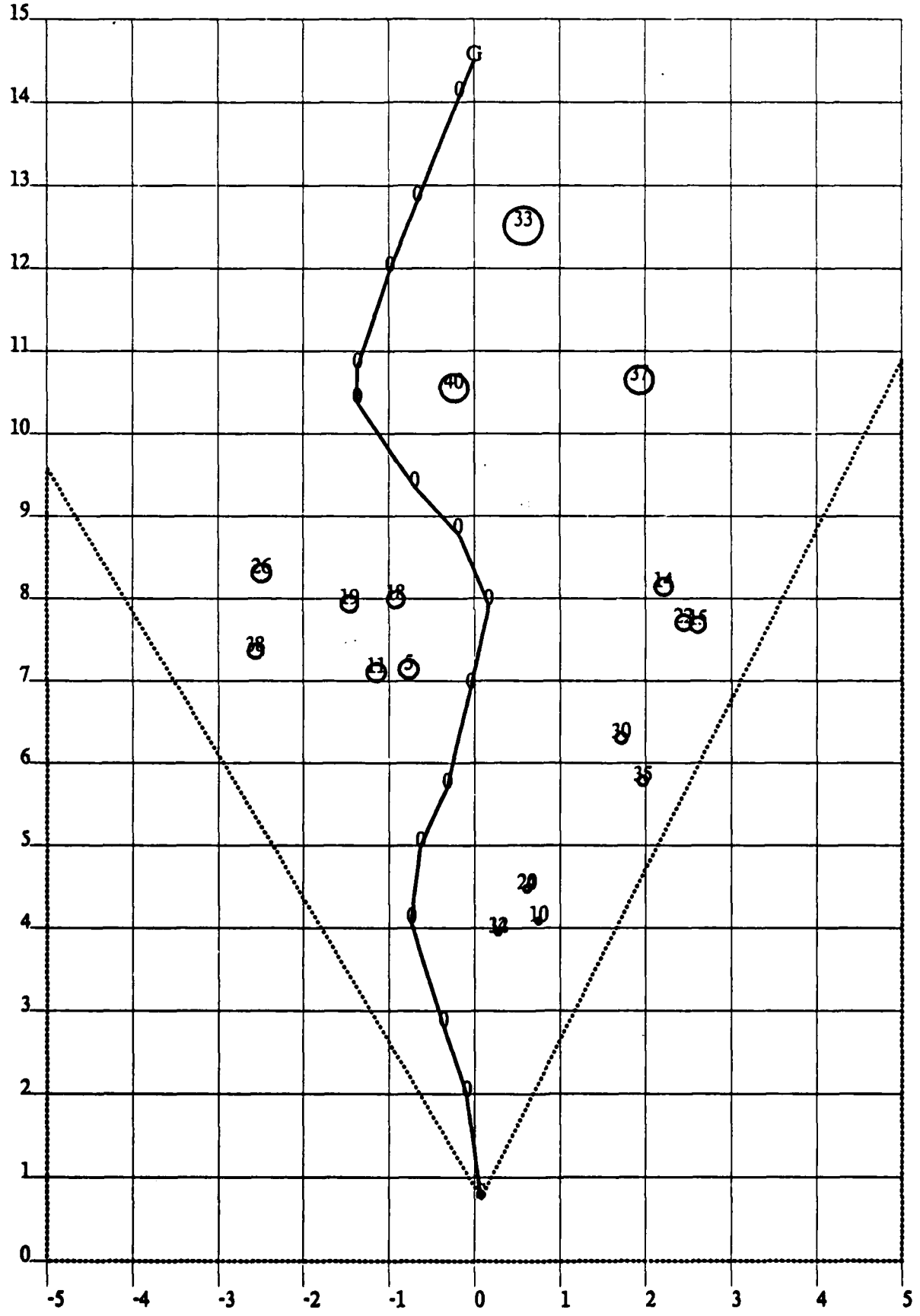


Figure 7

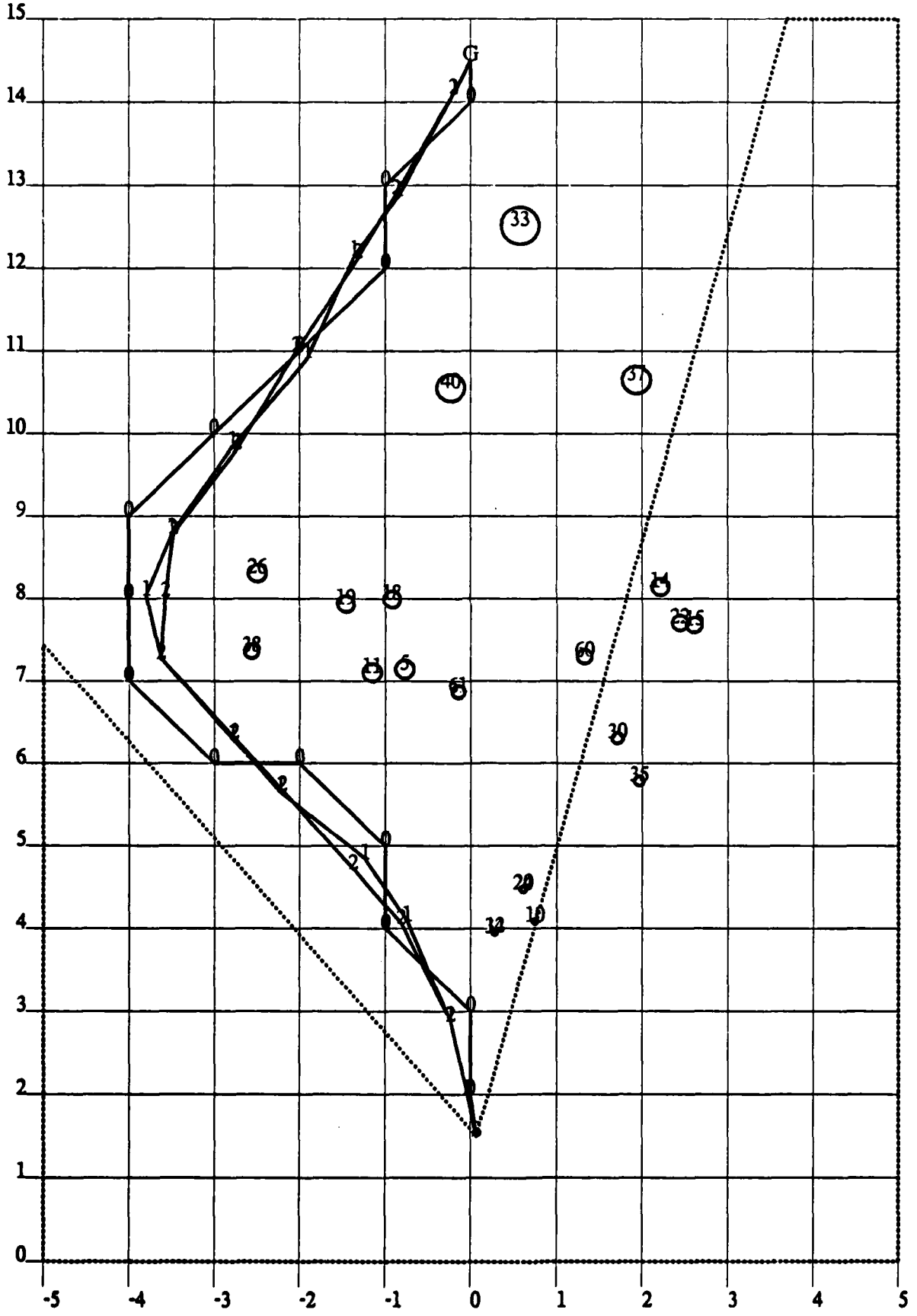


Figure 8

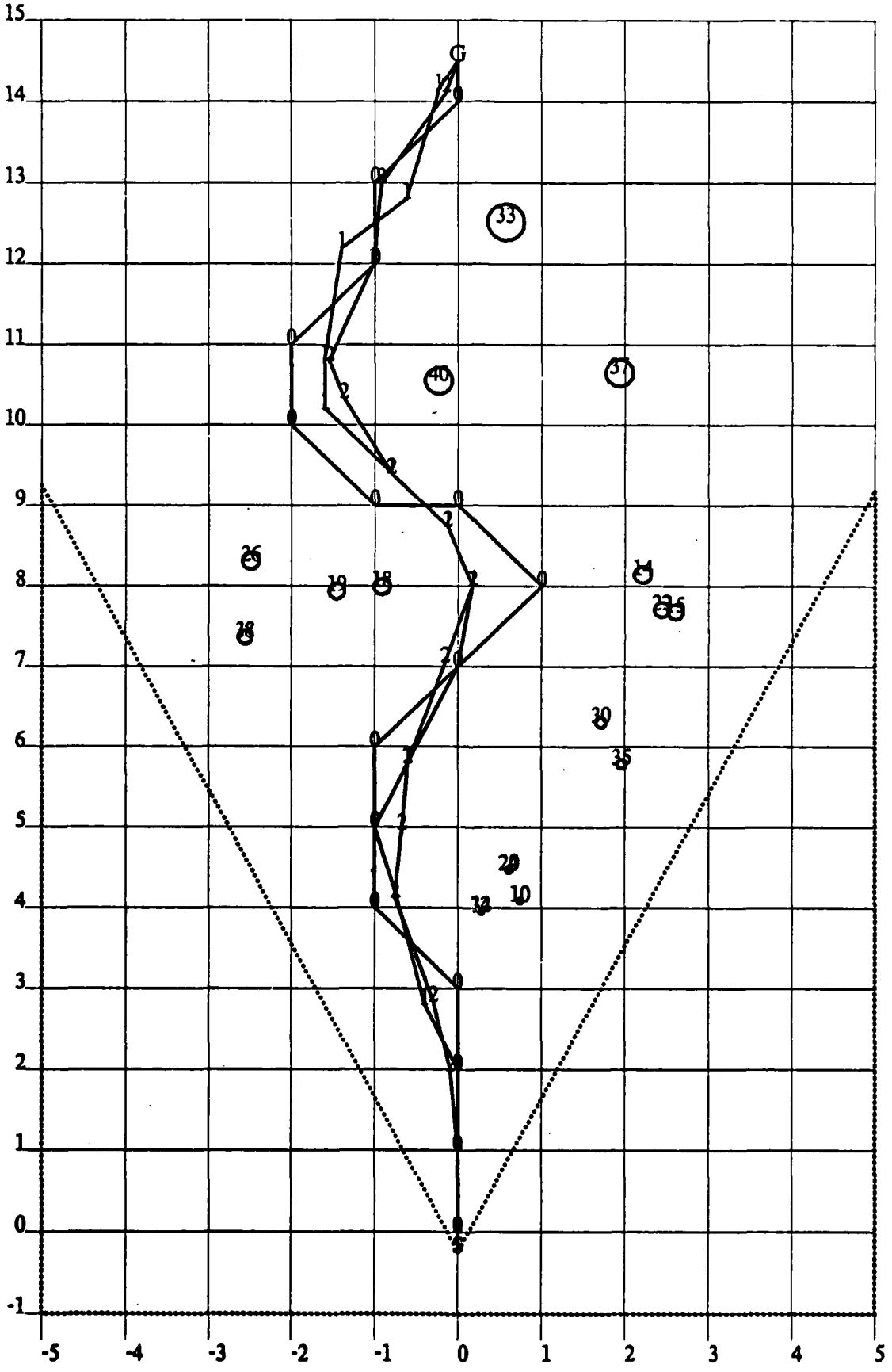


Figure 9

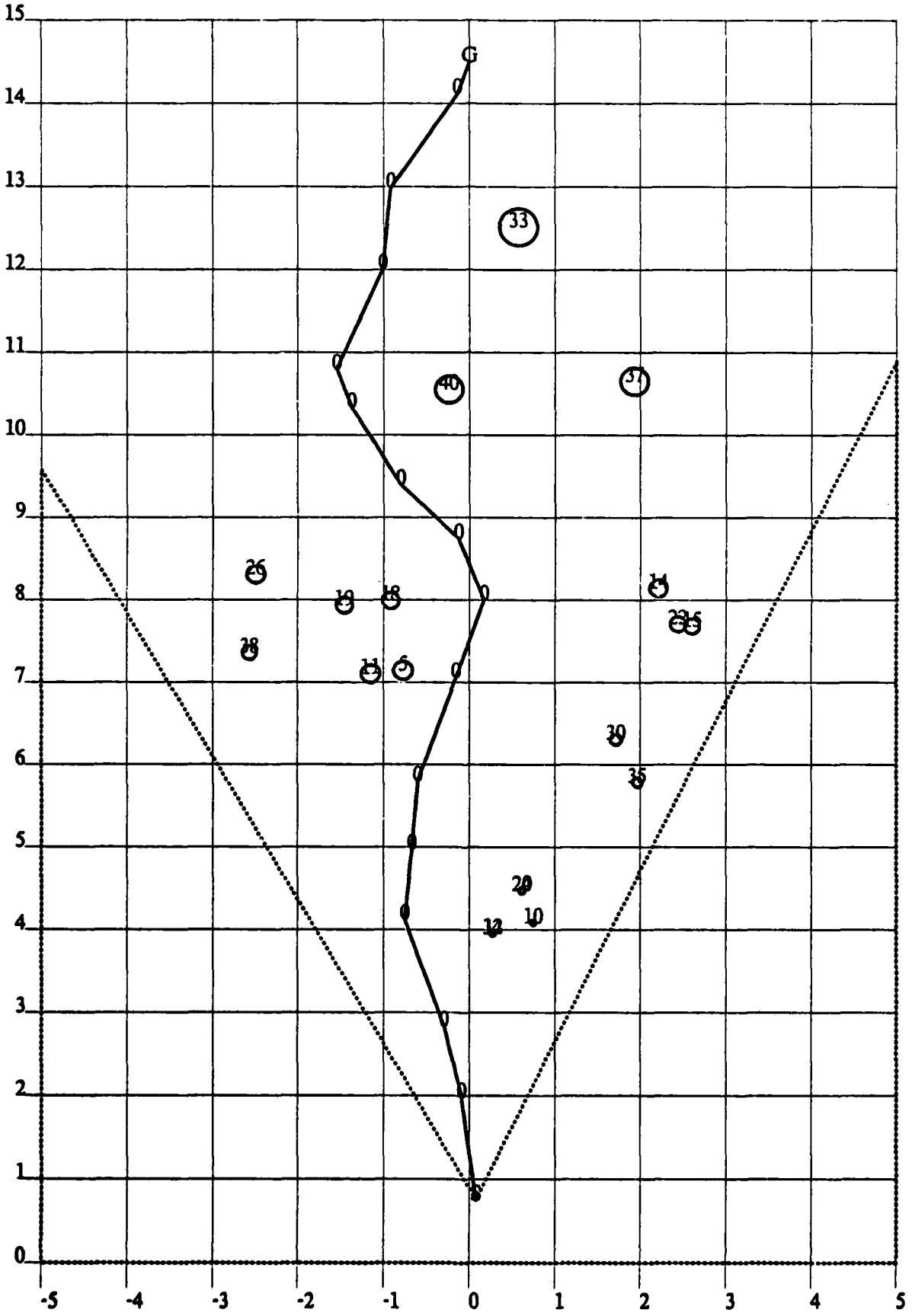
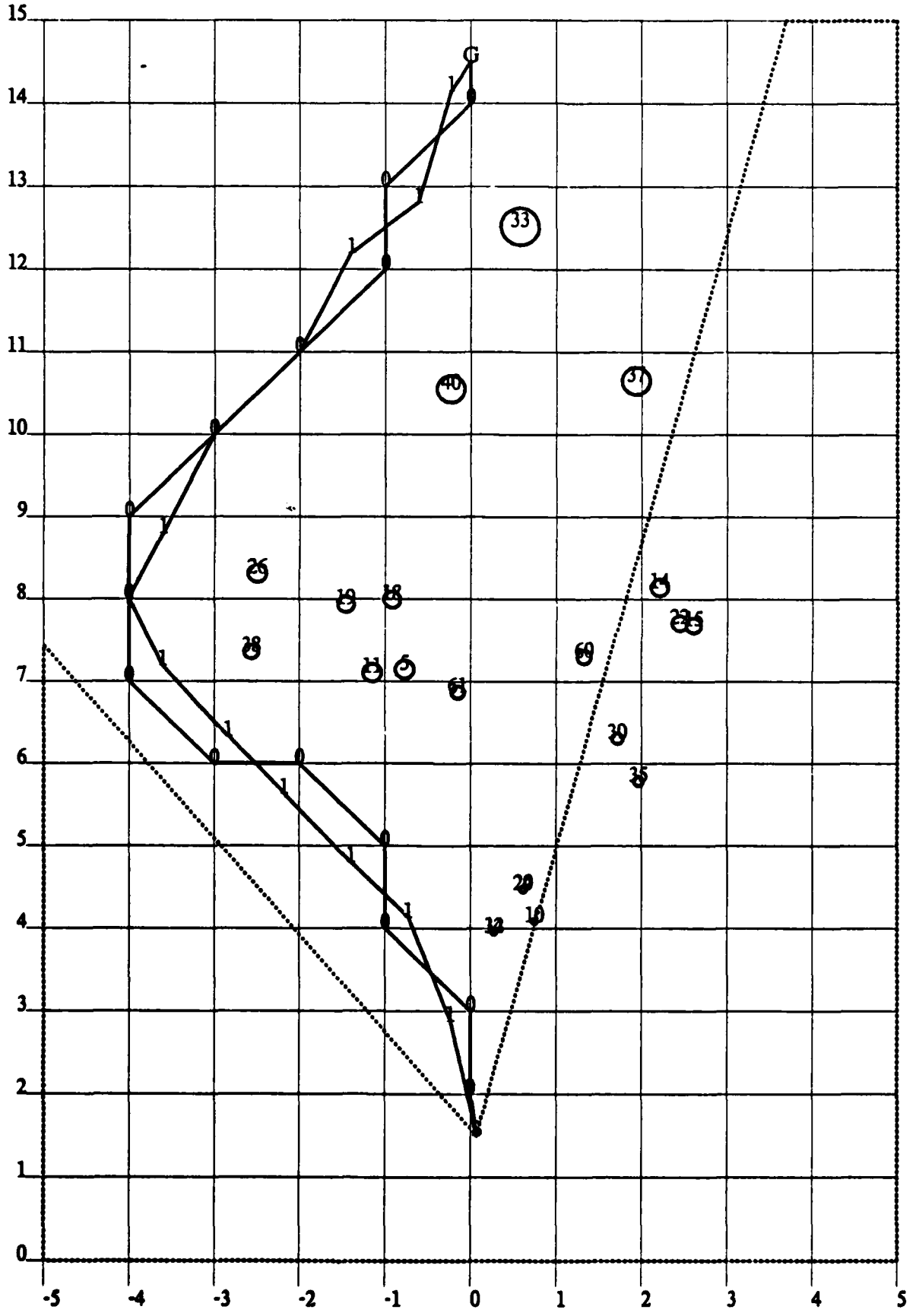

15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
-5
-4
-3
-2
-1
0
1
2
3
4
5
G
33
40
37
26
19
18
28
11
6
14
23
25
30
35
20
10
12

Figure 10



END

FILMED

DTIC